CS 163 Filtering Lab

Fall 2023

October 8, 2023

1 Complex Number Refresher

1.1 The Imaginary Number i

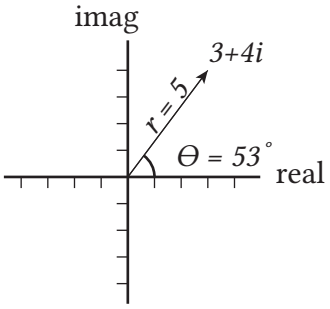
You might remember from high school algebra class the imaginary number $i = \sqrt{-1}$. This number is often used to find the roots of polynomials. For example, suppose we have a polynomial $y = x^2 + 1$. We can use the quadratic equation to find its roots:

$$y = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{\pm\sqrt{-4(1)(1)}}{2} = \frac{2\sqrt{-1}}{2} = \sqrt{-1} = i$$

1.2 Complex Numbers

Complex numbers have both an imaginary component and a real component. The real and imaginary parts are orthogonal, meaning that changing the value of one component doesn't affect the other.

Example Consider the complex number $3 + 4i$. The figure below shows that number drawn as a vector in the complex plane. The complex plane is like a Cartesian coordinate system where the horizontal dimension is the real axis and the vertical dimension is the imaginary axis.



There are two common ways that we can represent numbers in the complex plane. The first way is to just simply write out the real and imaginary components, as in $3 + 4i$. The second way is using polar coordinates, where we compute the radius of the complex vector and the angle it makes with the real axis. In the example of $3 + 4i$, we compute the radius using the Pythagorean theorem:

$$r = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

We can compute the angle it makes with the real axis using the sine function:

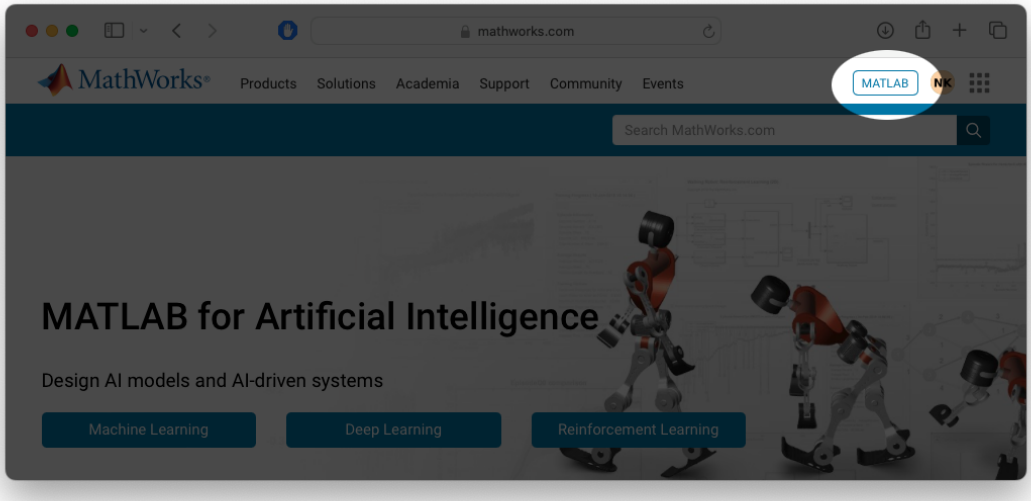
$$sin(\theta) = \frac{4}{5}$$

$$\theta = asin(\frac{4}{5}) \approx 53^\circ$$

Pro Tip The length of a complex vector is often called its *magnitude*, and in MATLAB it can be computed using the `abs()` function. The angle that a complex vector makes with the real axis is called its *phase*, and it can be computed in MATLAB using the `angle()` function.

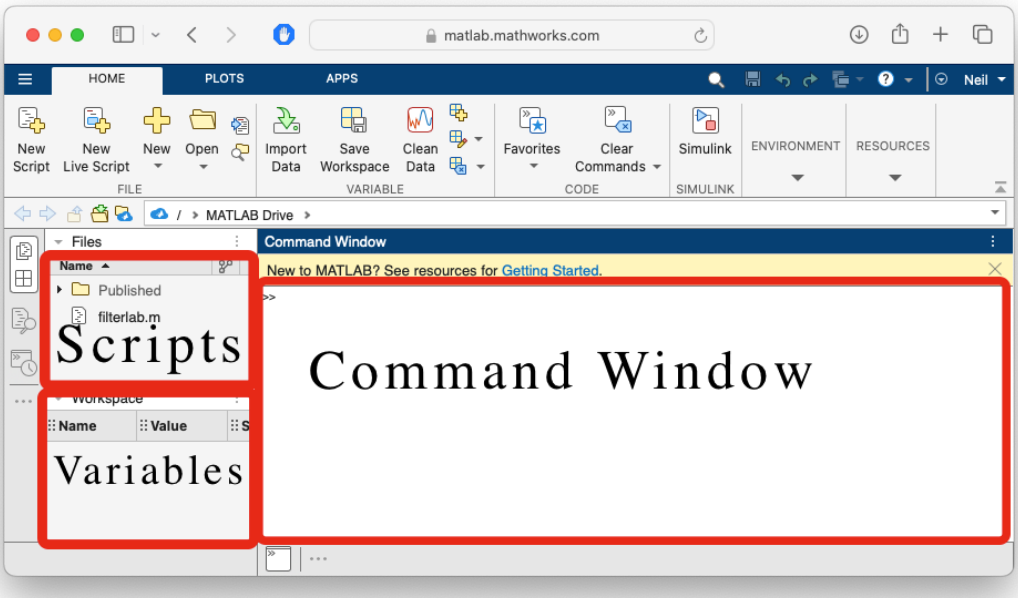
2 Basic MATLAB

Start by opening a web browser to `https://matlab.mathworks.com` and creating an account with your LUC email address. Once you are logged in, open up the online MATLAB UI by clicking the button in the upper right corner of the page:



This should give you a blank MATLAB command window (shown below). In the command section of the IDE, you can type MATLAB commands. Any variables you create will show up in the variables panel to the left. You can also create scripts, which will show up in the scripts panel to the left. To create a script, click the "New Script" button in the upper left corner of the window.

Type the following code either in a script or in the command window.




```
% Set the number of samples in each signal vector
N = 300;

% Create a vector of equally-spaced sample times in the range [0,16 pi]
t = linspace(0,16*pi,N);

% Compute the sin() of every element in vector t, storing the result in vector s
s = sin(t);

% Plot s(t)
plot(t,s);
```

The `linspace()` function creates a vector `N` equally-spaced points between 0 and 16π.

The `sin()` function computes the sine of either a scalar value or every element in a vector. In our program, vector `t` consists of 300 equally spaced points in time. `sin(t)` returns the a vector of 300 points, each of which is the sine of the corresponding point in vector `t`.

Pro Tip If you need help with a MATLAB function, you can type `doc XXX` into the MATLAB command window (where `XXX`) is the function name. For example, to pull up documentation for the `linspace()` function:

```
>> doc linspace
```

Semicolons in MATLAB In MATLAB, semicolons are optional. A semicolon can be used to terminate a line of code, but it is not required. Lines of code ending with a semicolon will not print the output to the command window. Lines of code that do not end with a semicolon do print output.

```
>> N = 300;
>> N = 300

N =

```

```
300
```

Even in scripts, lines that don't end in a semicolon will print output to the command terminal when they run.

3 Adding Noise

We want to add random noise to our signal `s` that we generated above. MATLAB has several functions for producing vectors of random numbers. Create a new $1 \times N$ vector called `n` using one of the functions below with random numbers.

`rand()` generates uniformly distributed random number on the interval $(0, 1)$. `rand(n)` generates an $n \times n$ matrix of uniformly distributed random numbers on the interval $(0, 1)$. `rand([1 n])` generates a $1 \times n$ vector of uniformly distributed random numbers on the interval $(0, 1)$.

`randn()` generates normally distributed random number taken from the standard normal distribution ($\mu = 0, \sigma = 1$). `randn(n)` generates an $n \times n$ matrix of normally-distributed random numbers from the standard normal distribution. `randn([1 n])` generates a $1 \times n$ vector of normally-distributed random numbers from the standard normal distribution.

3.1 Adding Noise to the Signal

After creating your noise vector `n`, add it to to your clean signal to create the measured signal `m`. Plot the measured signal `m`.

```
m = s + n;
plot(t,m);
```

Pro Tip: Each time you call the `plot()` function, it will draw over the previous plot. If you want to keep the previous plot and generate a new figure, you can use the `figure()` command:

```
figure(1);
plot(t,s);
figure(2);
plot(t,m);
```

4 Building an Orthogonal Matrix for the Filter

Next, we are going to build an orthogonal matrix that we will use to apply the filter using MATLAB's `dftmtx()` function. `dftmtx()` takes one parameter `N`, the size of the matrix to be produced. It returns an `N x N` orthogonal matrix `Q` which contains the complex-valued[1] Fourier basis.

```
Q = dftmtx(300);
```

When we multiply a vector by `Q`, we are rotating it into the Fourier basis. Unfortunately for us, vectors in the Fourier basis are complex-valued (because the Fourier basis is complex valued), which makes them difficult to visualize. Normally, when we visualize complex-valued vectors in the Fourier basis, we plot the magnitude of each component.

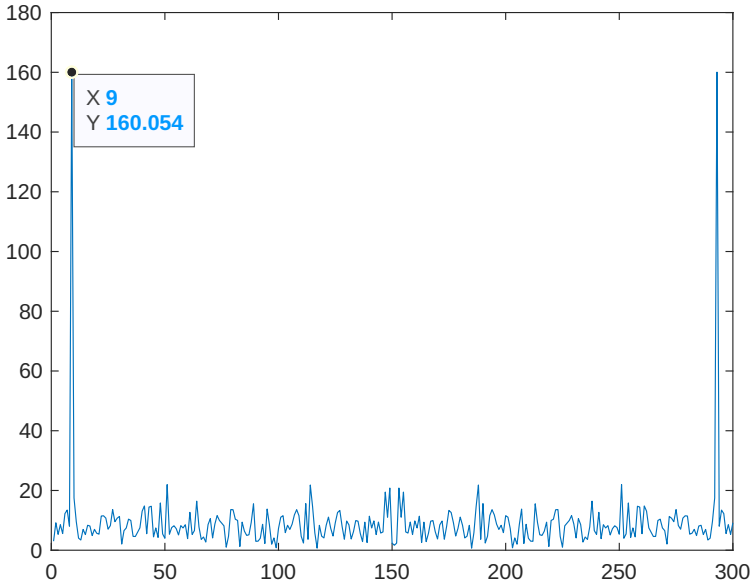
4.1 Rotating the Noisy Signal into the Fourier Basis

[1] Each element of the `Q` matrix has both real and imaginary parts.

```
% Apply the coordinate rotation by multiplying Q by vector s.
% NOTE: s is a row vector, We must take its transpose before multiplying by Q.
% In MATLAB, use a single quote to take the transpose of a vector or matrix.
% s' means transpose of s.
f = Q*m';
plot(abs(f))
```

The above code will plot the magnitudes of each component of the complex-valued vector `f`. In my plot (shown below) there are two components that have a large magnitude: one at index 9 and another at index 293. I found these by hovering my mouse over the plot.

These two components are the signal (the sine wave `s` we made at the beginning). All the other components with small magnitude are noise.



To remove the noise from our signal, we will just set all the non-signal components to zero. This has the effect of projecting the noisy vector `f` onto the dimension spanned by our signal. After applying the projection, we can invert the coordinate transform by multiplying `f` by Q^T. In MATLAB, take the transpose of a matrix or vector by putting a single quote after the variable name.

```
% Project the noisy vector into the signal dimension
f(10:292) = zeros(size(f(10:292)));

% Invert the coordinate rotation.
filtered_s = real(Q'*(f));
```

Plot your filtered signal and demo to Neil.

5 Filtering Audio

Download `blueskies.wav` from the course website. This is a grainy audio recording from the 1920s that we're going to be trying to filter. Drag the file into the files section of your MATLAB window to upload it to MATLAB. You can import the audio data into a MATLAB array using the `audioread()` function:

```
[a fs] = audioread('blueskies.wav');
```

`audioread()` returns two variables: `a` is the audio data from the file and `fs` is the sampling frequency in Hertz (samples per second). Since the WAV file is in stereo, we can extract only the left channel:

```
a = a(:,1);
```

Our audio vector `a` contains a few million samples. We will process only the first hundred thousand samples (just over 2 seconds of audio).

```
s = a(1:100000);
```

Process the audio in the same way as you did above by applying matrix transformation Q then zeroing out the low-amplitude vector components. After filtering the rotated vector, transform it back by multiplying by Q^T. Use the `audiowrite()` MATLAB function to write out the filtered audio file. Download that filtered file and listen to it on your computer.

```
audiowrite('filtaudio.wav', s_filt, fs);
```